Integrating Conflict Driven Clause Learning to Local Search

Gilles Audemard Jean-Marie Lagniez Bertrand Mazure Lakhdar Saïs *

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{audemard,lagniez,mazure,sais}@cril.fr

This article introduces SATHYS (*SAT HYbrid Solver*), a novel hybrid approach for propositional satisfiability. It combines local search and conflict driven clause learning (CDCL) scheme. Each time the local search part reaches a local minimum, the CDCL is launched. For SAT problems it behaves like a tabu list, whereas for UNSAT ones, the CDCL part tries to focus on minimum unsatisfiable sub-formula (MUS). Experimental results show good performances on many classes of SAT instances from the last SAT competitions.

1 Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is a central issue in many computer science and artificial intelligence domains, like theorem proving, planning, non-monotonic reasoning, VLSI correctness checking. These last two decades, many approaches have been proposed to solve large SAT instances, based on logically complete or incomplete algorithms. Both local-search techniques [29, 28, 18] and elaborate variants of the Davis-Putnam-Loveland-Logemann DPLL procedure [6] [27, 8], called modern SAT solvers, can now solve many families of hard SAT instances. These two kinds of approaches present complementary features and performances. Modern SAT solvers are particularly efficient on the industrial SAT category while local search performs better on random SAT instances.

Consequently, combining stochastic local search (SLS) and conflict driven clause learning (CDCL) solvers seems promising. Note that it was pointed as a challenge by Selman *et al.* [30] in 1997. Such methods should exploit the quality and differences of both approaches. Furthermore, the perfect hybrid method has to outperform both local search and CDCL solvers. A lot of attempts have been done last decade [4]. These different attempts will be discussed in section 3.

In this paper, we propose another hybridization of local search and modern SAT solver, named SATHYS (*SAT HYbrid Solver*). The local search solver is the main one. Each time it reaches a local minimum, the CDCL part is called and assigns some variables. This part of our solver is expected to have different behaviours depending on the kind of formula to solve. In case of a satisfiable one, the CDCL part can be seen as a tabu list [12] in order to protect good variables and avoid to reach the same minimum quickly. On the other hand, for unsatisfiable formulas, it tries to focus the search on minimum unsatisfiable sub-formulas (MUS) [9, 14, 15], allowing to concentrate on a small part of the whole formula. Like this, the CDCL component of SATHYS is used as a strategy to escape from local minimum.

The rest of the paper is organized as follows. Section 2 introduces different notions necessary for understanding the rest of the paper. Section 3 discusses different hybrid methods. Section 4 gives the

*supported by ANR UNLOC project ANR08-BLAN-0289-01

Y. Deville and C. Solnon (Eds): Sixth International Workshop
on Local Search Techniques in Constraint Satisfaction (LSCS'2009).
EPTCS 5, 2009, pp. 55–68, doi:10.4204/EPTCS.5.5

insights of our method. In section 5, we give the details and algorithms of SATHYS. Before a conclusion, section 6 provides different experiments.

2 Preliminary definitions and technical background

2.1 Definitions

Let us give some necessary definitions and notations. Let $V = \{x_1 ... x_n\}$ be a set of boolean variables, a literal ℓ is a variable x_i or its negation $\bar{x_i}$. A clause is a disjunction of literals $c_i = (\ell_1 \vee \ell_2 ... \vee \ell_{n_i})$. A unit clause is a clause with only one literal. A formula Σ is in conjunctive normal form (CNF) if it is a conjunction of clauses $\Sigma = (c_1 \wedge c_2 ... \wedge c_m)$. The set of literals appearing in Σ is denoted $\mathscr{V}_\Sigma$. An interpretation $\mathscr{I}$ of a formula Σ associates a value $\mathscr{I}(x)$ to variables in the formula. An interpretation is *complete* if it gives a value to each variable $x \in \mathscr{V}_\Sigma$, otherwise it is said *partial*. A clause, a CNF formula and an interpretation can be conveniently represented as sets. A *model* of a formula Σ, denoted $\mathscr{I} \models \Sigma$, is an interpretation $\mathscr{I}$ which satisfies the formula Σ i.e. satisfies each clause of Σ. Then, we can define the SAT decision problem as follows: is there an assignment of values to the variables so that the CNF formula Σ is satisfied?
Let us introduce some additional notations.

- The negation of a formula Γ is denoted $\bar{\Gamma}$
- $\Sigma_{|_\ell}$ denotes the formula Σ simplified by the assignment of the literal ℓ to true. This notation is extended to interpretations: Let $\mathscr{P} = \{\ell_1, ..., \ell_n\}$ be an interpretation, $\Sigma_{|_\mathscr{P}} = (...(\Sigma|_{\ell_1})...|_{\ell_n})$;
- Σ^* denotes the formula Σ simplified by unit propagation;
- $\models_*$ denotes logic deduction by unit propagation: $\Sigma \models_* l$ means that the literal x is deducted by unit propagation from Σ i.e. $(\Sigma \wedge \bar{\ell})^* = \bot$. One notes $\Sigma \models_* \bot$ if the formula is unsatisfiable by unit propagation.
- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the opposite literal $\neg x$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. A resolvent is called *tautological* when it contains opposite literals.

2.2 Local Search Algorithms

Local search algorithms for SAT problems use a stochastic walk over complete interpretations of Σ. At each *step* (or *flip*), they try to reduce the number of unsatisfiable clauses (usually called a descent). The next complete interpretation is chosen among the neighbours of the current one (they differ only on one literal value). A local minimum is reached when no descent is possible. One of the key point of stochastic local search algorithms is the method used to escape from local minimum. For lack of space, we can not provide a general algorithm of local search solver. For more details, the reader will refer to [19].

2.3 CDCL solvers

Algorithm 1 shows the general scheme of a CDCL solver (due to lack of space, we can not provide details for all subroutines). A typical branch of a CDCL solver is a sequence of decisions, followed by propagations, repeated until a conflict is reached. Each decision literal (lines 18–20) is assigned at a given decision level (dl), deducted literals (by unit propagation) have the same decision level. If all variables

Algorithm 1: CDCL solver

```
   Input: a CNF formula Σ
   Output: SAT or UNSAT
 1 I = ∅ ;                                          /* interpretation */
 2 dl = 0 ;                                         /* decision level */
 3 x_c = 0 ;                                    /* number of conflicts */
 4 while (true) do
 5 |  γ = BCP(Σ,I);
 6 |  if (γ!=null) then
 7 |  |  x_c = x_c + 1;
 8 |  |  β =conflictAnalysis(Σ,I,c);
 9 |  |  bl = computeBackjumpingLevel(γ,I);
10 |  |  if (bl < 0) then return UNSAT;
11 |  |  Σ = Σ ∪ {γ};
12 |  |  if (restart()) then bl = 0;
13 |  |  backjump(Σ,I,bl);
14 |  |  dl = bl;
15 |  else
16 |  |  if (all variables are instanciated) then
17 |  |  |  return SAT;
18 |  |  ℓ = chooseDecisionLiteral(Σ);
19 |  |  dl = dl + 1;
20 |  |  I = I ∪ {ℓ};
21 |
22 end
```

are assigned, then $\mathscr{I}$ is a model of Σ (lines 16–17). Each time a conflict is reached by unit propagation (then γ is the conflict clause) A nogood β is computed (line 8) using a given scheme, usually the first-UIP (Unique Implication Point) one [31] and a backjump level is computed. At this point, It may have proved the unsatisfiability of the formula Σ. If it is not the case, the nogood β is added to the clause database and backjump is done (lines 11–14). Finally, sometimes CDCL solvers enforce restarts (different strategies are possible [20]). In this case, one backjump in the top of the search tree.

2.4 Muses

Minimum unsatisfiable sub-formulas (MUS) of a CNF formula represent the smallest explanations for the inconsistency in term of the number of clauses. MUS are very important in order to circumscribe and highlight the source of contradiction of a given formula. Formally, one has:

Definition 1 *Let Σ be a CNF formula. A MUS Γ of Σ is a set of clauses such that:*

1. $\Gamma \subseteq \Sigma$;
2. Γ *is unsatisfiable;*
3. $\forall \Delta \subset \Gamma, \Delta$ *is satisfiable.*

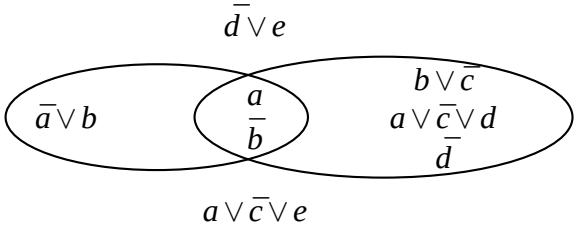

Figure 1: All MUS of the formula Σ (example 1)

Example 1 *Let* $\Sigma = \{\bar{d} \vee e,\ b \vee \bar{c},\ \bar{d},\ \bar{a} \vee b,\ a,\ a \vee \bar{c} \vee e,\ \bar{a} \vee c \vee d,\ \bar{b}\}$ *be a CNF formula. Figure 1 represents all MUS of* Σ*.*

Due to unsatisfiability of MUS, one has the following property:

Proposition 1 *Let* Σ *be an unsatisfiable CNF formula,* Γ *a MUS of* Σ*.*

$$\forall \mathscr{I} \text{ an interpretation over } \mathscr{V}_\Sigma,\ \exists c \in \Gamma \text{ such that } \mathscr{I} \not\models c$$

Let us consider a CNF formula Σ and a complete interpretation $\mathscr{I}_c$. We say that the literal ℓ satisfies (resp. falsifies) a clause $\beta \in \Sigma$ if $\ell \in \mathscr{I}_c \cap \beta$ (resp. $\ell \in \mathscr{I}_c \cap \bar{\beta}$). We note $\mathscr{L}^+_{\mathscr{I}_c}(\beta)$ (resp. $\mathscr{L}^-_{\mathscr{I}_c}(\beta)$), the set of literals satisfying (resp. falsifying) a clause β. The following definitions were introduced in [13].

Definition 2 (once-satisfied clause) *A clause* β *is said* once-satisfied *by an interpretation* $\mathscr{I}_c$ *on literal* z *if* $\mathscr{L}^+_{\mathscr{I}_c}(\beta) = \{z\}$.

Definition 3 (critical and linked clauses) *Let* $\mathscr{I}_c$ *be a complete interpretation. A clause* α *is critical wrt* $\mathscr{I}_c$ *if* $|\mathscr{L}^+_{\mathscr{I}_c}(\alpha)| = 0$ *and* $\forall \ell \in \alpha,\ \exists \alpha' \in \Sigma$ *with* $\bar{\ell} \in \alpha'$ *and* $|\mathscr{L}^+_{\mathscr{I}_c}(\alpha')| = 1$*. Clauses* α' *are linked to* α *for the interpretation* $\mathscr{I}_c$*.*

Example 2 *Let* $\Sigma = (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c}) \wedge (c \vee \bar{a})$ *be a formula and* $\mathscr{I}_c = \{a, b, c\}$ *an interpretation. The clause* $\alpha_1 = (\bar{a} \vee \bar{b} \vee \bar{c})$ *is critical. The other clauses of* Σ *are linked to* α_1 *for* $\mathscr{I}_c$*.*

The following properties was proposed and exploited in order to compute MUS by [13].

Proposition 2 *In a minimum (local or global), the set of falsified clauses are critical.*

Proposition 3 *In a minimum (local or global), at least one of clause of each MUS is critical.*

3 Related Works

As it was suggested in the introduction, a lot of different approaches have been proposed to combine local search and DPLL based ones. One can divide such hybridizations in three different categories depending on the kind of the main solver. First, the main solver can be the SLS one. In that case, DP is used in order to help SLS [26, 5, 1, 17]. All of these approaches use the local search component as an assistance for the heuristic choice for variable assignment. Some of them try to focus the search on the unsatisfiable part of the formula [26], others on the satisfiable one [2, 1]. Furthermore, this step can be achieved before the search [5] or dynamically at each decision nodes [26].

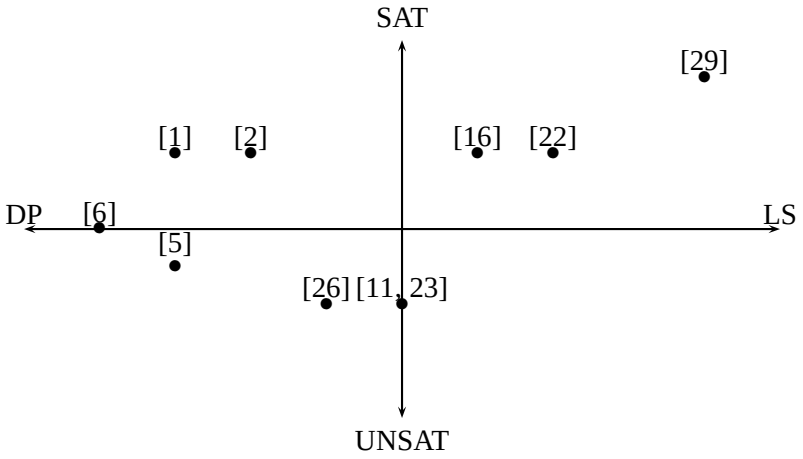

Figure 2: Classification

The second category of hybridizations is the opposite, that is, the SLS solver is the core of the method and the DPLL one helps it [16, 22]. In [16], the DPLL solver is used in order to find dependencies between variables. Then, the local search framework is called on a subset of variables (the independent ones). Whereas, in [22] (note that this method is for constraint satisfaction problems), the local search engine is used to find a promising partial interpretation.

Finally, the last category contains hybrid solvers where the both engines work together [11, 23]. The second method is an improvement of the first one. The local search tries to find a solution. After some time, it stops and sends all falsified clauses by the current interpretation to the CDCL part. This last one has the responsibility to find a model to this sub-formula. If it proves unsatisfiable, then the whole instance is unsatisfiable too.

We propose in Figure 2 a classification of all of these approaches. The X-axis corresponds to the kind of search. For example, DPLL is at the left, whereas walksat is at the right. The Y-axis corresponds to the ability to solve SAT and/or UNSAT formulas. Then, walksat is at the top of the classification. Methods introduced above are located in this graph. Of course, this classification is subjective and and it can be subject of discussion. It is here to help the reader to understand all of these approaches.

4 Intuition

In this section, we provide insights of our hybrid approach SATHYS. They are related to the satisfiability or unsatisfiability of the formulas. First note that the SLS engine is the core of our method. Then, the Local search part tries to find a solution. When a local minimum is reached, the CDCL part of the solver is launched. It works like a tabu list in case of satisfiable formula and tries to focus on MUS for unsatisfiable ones. Let us explain the main differences now.

4.1 SAT instances

Much research has been done on meta-heuristics. Among them, Tabu search was introduced in 1986 by Glover [12] and extended to the SAT case in 1995 [25]. The main idea of tabu search consists, in a given position (interpretation), in exploring neighbours and choosing as the next position the one which minimises the objective function.

It is crucial to note that such an operation could increase the objective function value: it is the case when all neighbours have a greater value. Then, this mechanism allows to escape from local minimum. However, the main drawback is that at the next step, one goes back in the same local minimum. To avoid this, heuristic needs memory for the last explored positions to be forbidden. These positions are *tabu*.

Already explored positions are stored in a queue (usually called *tabu list*) of a given length which is a parameter of the method. This list must contain complete positions, which can be prohibitive. To go round this, one can store only previous actions, associated to values of the objective function. The length parameter is very important. A lot of work have been done to provide optimal length, statically [26] or dynamically [3].

We propose to keep the set of tabu variables by using a partial interpretation computed with unit propagation engine. When a variable becomes tabu, it is assigned in the CDCL solver part and propagated. Then, resulting interpretation is used as a tabu list. There are two advantages: firstly, the length of the tabu list is dynamic, it depends of unit propagation and backjumping. Secondly, unit propagation allows to catch some functional dependencies in the tabu list.

4.2 UNSAT instances

First of all, note that if an instance is unsatisfiable then, whatever is the complete interpretation, a falsified clause exists. Furthermore, if an instance is unsatisfiable, then it contains at least one MUS. This MUS, i.e. a subset of clauses of the formula, is often smaller than the global formula and, then, can contain less variables. Then, in the case of unsatisfiable formula, it is advantageous to focus the search on such variables.

In the frame of MUS detection, Grégoire *et al.* [13, 15] shown that local search provides good heuristics, concerning inconsistent kernel detection. These methods use properties 2 and 3 in order to balance clauses which could be part of a MUS.

The proposed method in this paper is based on this principle. When a local minimum is reached, property 2 assures that the set of clauses falsified by current interpretation are critical. Given that such clauses could be part of a MUS, we choose one of them to make it totally true. Therefore only the variables of a kernel are expected to be taken into account.

5 Implementation

As explicated in the previous section, the core of our solver SATHYS is the local search component. It is based on an iterative search process that in each step moves from one point to a neighbouring one until discovering a solution. At each step it tries to reduce the number of falsified clauses. When it is not possible, a local minimum is reached. In that case, the CDCL part is called. It chooses a falsified clause and assigns all of its literals such that the clause becomes totally valid. All literals of the chosen clause are decision nodes. Of course unit propagation is achieved. In this manner, it escapes from the local minimum and the SLS part of the hybrid solver can be used again. Note that all variables assigned by the CDCL part are fixed and can not be flipped by the SLS solver. Of course, during the CDCL process, a conflict can occur. In that case, conflict analysis is performed, a clause is learnt and a backjump is done. Then, some of fixed variables become free and can be flipped again. This conflict analysis makes the solver able to prove unsatisfiability.

Algorithm 2 takes a CNF formula Σ in parameter and returns SAT or UNSAT. It is based on WSAT-like algorithms. Two variables are used. A complete interpretation $\mathscr{I}_c$ for the local search engine (ini-

Algorithm 2: SATHYS

```
   Input: Σ a CNF formula
   Result: SAT if Σ is satisfiable, else UNSAT
1  while (true) do
2      I_c ← Init(Σ);
3      I_p ← ∅;
4      for j ← 1 to MaxFlips do
5          if I_c ⊨ Σ_|I_p then
6              return SAT;
7          Γ = {α ∈ Σ_|I_p | I_c ⊭ α}          /* set of falsified clauses */;
8          while Γ ≠ ∅ do
9              α ∈ Γ;
10             if ∃x ∈ α allowing a descent then
11                 flip(x);
12                 break;
13             else
14                 Γ ← Γ \ {α};
15         if Γ = ∅ then                        /* local minimum */
16             α ∈ Σ_|I_p such that I_c ⊭ α;
17             if (fix(Σ, I_c, I_p, α)=UNSAT) then
18                 return UNSAT;
```

tialised randomly) and a partial interpretation $\mathscr{I}_p$ for the CDCL part (initialised to the empty set). In order to forbid to flip fixed literals by the CDCL part (the literals of $\mathscr{I}_p$), the SLS solver deals with $\Sigma_{|\mathscr{I}_p}$. If the current complete interpretation is a model of $\Sigma_{|\mathscr{I}_p}$ then SATHYS finishes and returns SAT (lines 5–6). Otherwise, if it exists a neighbour of $\mathscr{I}_c$ which allows to decrease the number of falsified clauses, it becomes the current complete interpretation (lines 8–14). If it is not the case, then a local minimum is reached (line 15). In that case, a falsified clause is randomly chosen and the function `fix` is called in order to fix new literals (lines 15–17). This function is explained below. It modifies interpretations $\mathscr{I}_c$ and $\mathscr{I}_p$ by fixing new variables and (if a conflict occurs during boolean propagation) freeing other ones. At this step, the CDCL solver can prove the unsatisfiability. Of course, if it is the case the search is done (line 17–18).

This whole process is repeated a given number of times (*MaxFlips*, line 4). After that, the solver tries to go in another area of the search space. Then, the process can continue until finding an answer.

Function `fix` is described in Algorithm 3. It works like a very simple CDCL solver. It takes a clause α in input. It takes also in input the complete interpretation $\mathscr{I}_c$ and the partial one $\mathscr{I}_p$ and modifies them. It returns *UNSAT* if the unsatisfiability is proven, and *UNKNOWN* otherwise. The main goal of this function is to fix new variables. To achieve this, it tries to totally satisfy the clause α. First of all, the set of decision denoted $\mathscr{E}$ is initialized. Whenever it is not empty and a conflict does not occur, a new decision variable is chosen and added to the partial interpretation and boolean unit propagation (BCP) is performed (lines 3–6). If a conflict occurs, then the process is stopped. A conflict analysis is done and the partial interpretation is repaired (`backjumping`). At this step the unsatisfiability can be

Algorithm 3: fix

Input: α a clause
Output: Σ a CNF, $\mathscr{I}_c$ a complete interpretation, $\mathscr{I}_p$ a partial interpretation
Result: $UNSAT$ if unsatisfiable is proven, $UNKNOWN$ otherwise

```
1  γ ← ∅;
2  ℰ ← {x| x̄ ∈ α};
3  while (ℰ ≠ ∅) and (α = ∅) do
4  |  ℐ_p ← ℐ_p ∪ {x} tel que x ∈ ℰ;
5  |  γ ← BCP();
6  |  ℰ ← ℰ \ {x ∈ ℰ| x ∈ ℐ_p or x̄ ∈ ℐ_p};
7  if γ ≠ ∅ then
8  |  β =conflictAnalysis(Σ,ℐ_p,γ);
9  |  bl =computeBackjumpingLevel(γ,ℐ_p);
10 |  if (bl < 0) then return UNSAT;
11 |  Σ ← Σ ∪ {β};
12 ρ ← {x ∈ ℐ_c| x̄ ∈ ℐ_p};
13 ℐ_c ← ℐ_c \ {x̄| x ∈ ρ} ∪ ρ;
14 return UNKONWN;
```

proved. Otherwise, the obtained nogood is added to the clause database (lines 7–11). Then, the complete interpretation $\mathscr{I}_c$ is updated with the help of the partial one (note that $\mathscr{I}_p$ and $\mathscr{I}_c$ can not differ).

6 Experiments

Experimental results reported in this section were obtained on a Xeon 3.2 GHz with 2 GByte of RAM. The CPU time is limited to 1200 seconds.

Our approach is compared with:

- three SLS methods:
  1. classical WSAT [29], i.e. using random walk strategy
  2. RSAPS [21]
  3. ADAPTG2 [24]
- two recent hybrid methods submitted at the last SAT competition in 2009:
  1. HYBRIDGM [2]
  2. HYBRID1 [24]
- and two complete methods:
  1. CLS a local search method completed by adding resolution process [10]
  2. MINISAT [8] a well-known CDCL solver.

Instances used are taken from the last SAT competitions (`www.satcompetition.org`). They are divided into three different categories: crafted (1439 instances), industrial (1305) and random (2172). All instances are preprocessed with SatElite [7].

	Crafted		Industrial		Random	
	sat	unsat	sat	unsat	sat	unsat
ADAPTG2	326	0	232	0	1111	0
RSAPS	339	0	226	0	1071	0
WSAT	259	0	206	0	1012	0
CLS	235	75	227	102	690	0
SATHYS	322	191	466	309	341	14
HYBRIDGM	290	0	209	0	1114	0
HYBRID1	329	0	277	0	1126	0
MINISAT	402	369	588	414	609	315

Table 1: SATHYS versus some other SAT solvers

Table 1 summarizes the obtained results on this large number of instances. For more details on this experimental part, the reader can refer to `http://www.cril.fr/~lagniez/sathys`. For each category and for each solver we report the number of solved instances. Of course, MINISAT a state-of-the-art CDCL based complete solver is only considered to mention the gap between local search based techniques and complete modern SAT solvers on industrial and crafted instances. On random satisfiable instances, local search techniques generally outperform complete techniques.

Before analysing more precisely the table of results (Table 1), remark that only three solvers are able to solve unsatisfiable instances (MINISAT, CLS and SATHYS). The recent hybrid methods submitted at the last SAT competition cannot prove inconsistency in the allowed time.

On the crafted instances, SATHYS is very competitive and solves approximately the same number of satisfiable instances as RSAPS, ADAPTG2 and the recent hybrid methods. Furthermore, SATHYS solves much more instances than WSAT, its built-in solver. Concerning unsatisfiable crafted instances, as expected our approach is less efficient than MINISAT but it is proved highly more efficient than CLS.

Concerning industrial instances, SATHYS solves two times more satisfiable instances than SLS and hybrid methods. Once again, on unsatisfiable industrial instances, your solver is better than CLS but less efficient than MINISAT.

These results show that conflict analysis allows to solve efficiently structured SAT and UNSAT instances.

Finally, for the random category, we can note that SATHYS is unable to solve unsatisfiable problems. As pointed by MINISAT results, learning is not the good approach to solve random instances. As a summary, unfortunately our approach cannot reach the minisat performance. However the solver SATHYS is much more efficient than local search based algorithms and hybrid methods. It significantly improves WSAT, its built-in solver. Even if MINISAT is the best solver on crafted and industrial instances, these first results are very encouraging and reduce the gap between local search based techniques and DPLL-like complete solvers.

The figures 3, 4 and 5 give the classical cactus plot. For each tested method, the X-axis corresponds to the number of formulas and the Y-axis corresponds to the time needed to solve them if they were ran in parallel. When a method does not appear in the curve, that means that this method is not able to solve instance of this instances category. In these figures, we have distinguished satisfiable and unsatisfiable instances for each categories.

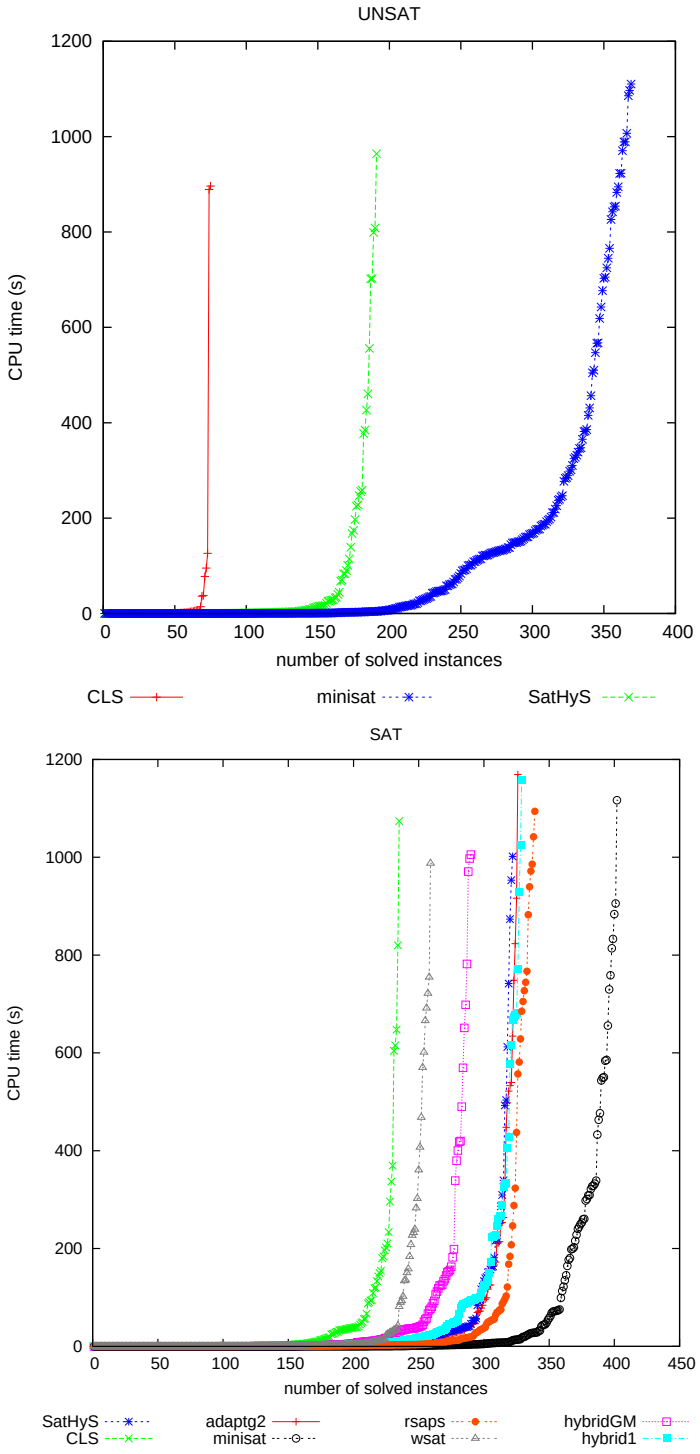

Figure 3: Crafted instances

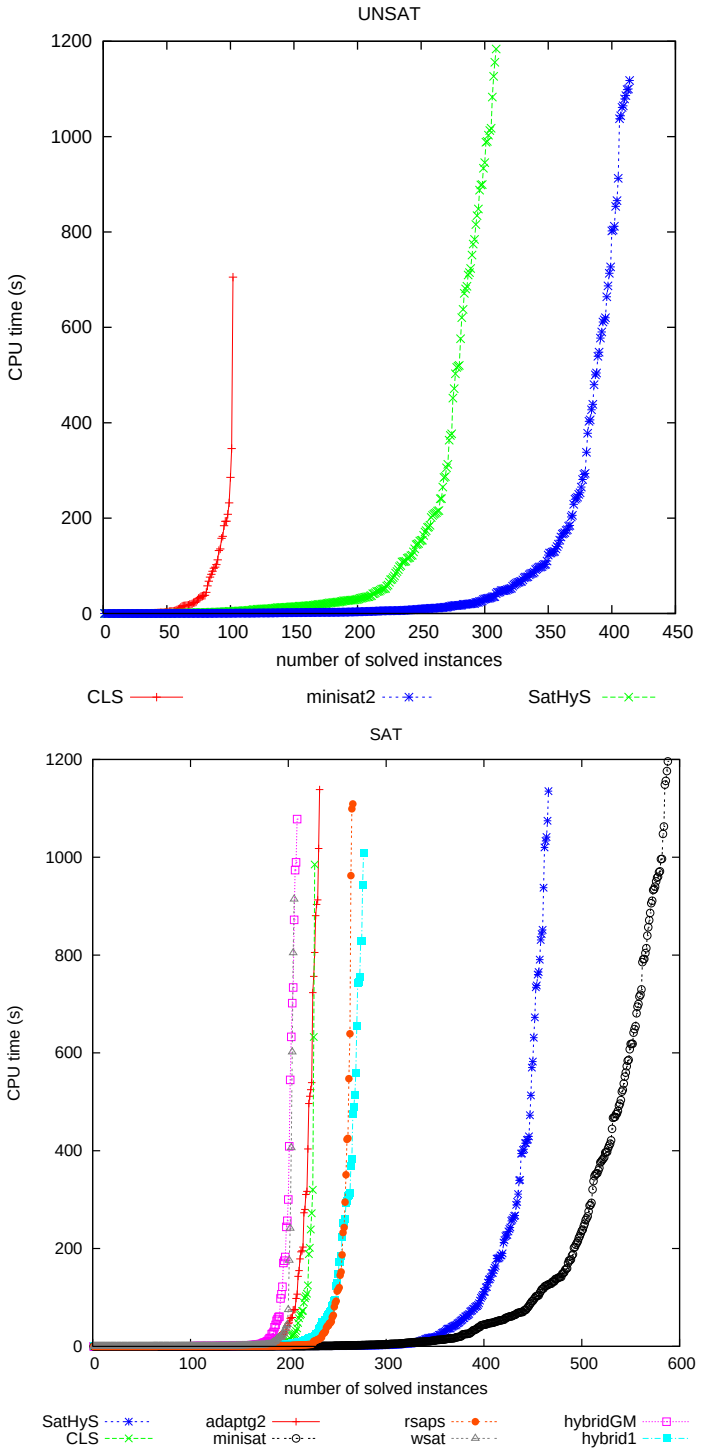

Figure 4: Industrial instances

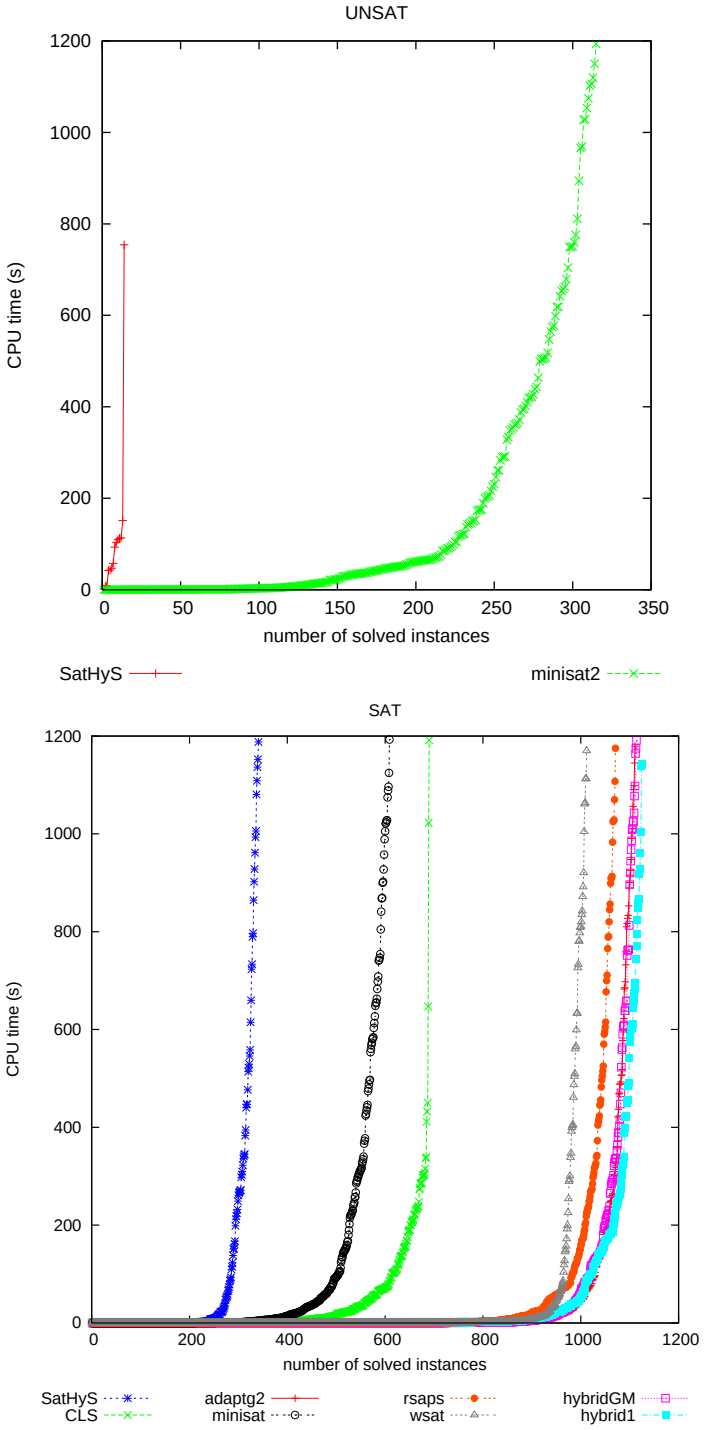

Figure 5: Random instances

7 Conclusion

In this paper a new integration of local search and CDCL based SAT solvers is introduced. This hybrid solver represents an original combination of both engines. The CDCL component can be seen as a new strategy for escaping from local minimum. This is achieved by the assignment of opposite literals from the falsified clause. In the case of satisfiable SAT instances, such assignments are supposed to behave like a tabu search approach, whereas for unsatisfiable ones, they try to focus on a small sub-part of the formula, which is minimally unsatisfiable (MUS). SATHYS, the resulting method, obtains very good results for a large category of instances. This new method can be improved in different ways. As it was pointed in the experimental section, our solver allows for more diversification and less intensification. First attempts have been done to correct this. Finally, we aim at designing a solver which would focuses only on an approximation of the MUS.

References

[1] B. B. & Fröhlich (2004): *WalkSAT as an Informed Heuristic to DPLL in SAT Solving.* Technical Report, CSE 573 : Artificial Intelligence.

[2] A. Balint, M. Henn & O. Gableske (2009): *A novel approach to combine a SLS- and DPLL-solver for the satisfiability problem*. In: *proceedings of SAT*.

[3] R. Battiti & M. Protasi (1997): *Reactive search, a history-sensitive heuristic for MAX-SAT*. *J. Exp. Algorithmics* 2, p. 2.

[4] A. Biere, M. Heule, H. van Maaren & T. Walsh, editors (2009): *Handbook of Satisfiability*, *Frontiers in Artificial Intelligence and Applications* 185. IOS Press.

[5] J. Crawford (1996): *Solving Satisfiability Problems Using a Combination of Systematic and Local Search*. In: *Second Challenge on Satisfiability Testing organized by Center for Discrete Mathematics and Computer Science of Rutgers University*.

[6] M. Davis, G. Logemann & D. Loveland (1962): *A machine program for theorem-proving*. *Communication of ACM* 5(7), pp. 394–397.

[7] N. Eén & A. Biere (2005): *Effective Preprocessing in SAT Through Variable and Clause Elimination*. In: *proceedings of SAT*. pp. 61–75.

[8] N. Een & N. Sörensson (2003): *An Extensible SAT-solver*. In: *proceedings of SAT*. pp. 502–518.

[9] E.Gregoire, B. Mazure & C. Piette (2006): *Tracking MUSes and Strict Inconsistent Covers*. In: *Sixth ACM/IEEE International Conference on Formal Methods in Computer Aided Design(FMCAD'06)*. San Jose (USA), pp. 39–46.

[10] H. Fang & W. Ruml (2004): *Complete Local Search for Propositional Satisfiability*. In: *proceedings of AAAI*. pp. 161–166.

[11] L. Fang & M. Hsiao (2007): *A new hybrid solution to boost SAT solver performance*. In: *proceedings of DATE*. pp. 1307–1313.

[12] F. Glover (1989): *Tabu search - Part I. ORSA Journal of Computing* , pp. 190–206.

[13] E. Gregoire, B. Mazure & C. Piette (2006): *Extracting MUSes*. In: *proceedings of ECAI*. pp. 387–391.

[14] E. Gregoire, B. Mazure & C. Piette (2007): *Boosting a Complete Technique to Find MSS and MUS thanks to a Local Search Oracle*. In: *International Joint Conference on Artificial Intelligence(IJCAI'07)*. Hyderabad (India), pp. 2300–2305.

[15] E. Gregoire, B. Mazure & C. Piette (2007): *Local-Search Extraction of MUSes*. *Constraints* 12(3), pp. 325–344.

[16] D. Habet, C.M. Li, L. Devendeville & M. Vasquez (2002): *A Hybrid Approach for SAT*. In: *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*. pp. 172–184.

[17] W. Havens & B. Dilkina (2004): *A Hybrid Schema for Systematic Local Search*. In: *Canadian Conference on AI*. pp. 248–260.

[18] E. Hirsch & A. Kojevnikov (2005): *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination*. *Annals of Mathematical and Artificial Intelligence* 43(1), pp. 91–111.

[19] H. Hoos & T. Stützle (2004): *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier.

[20] J. Huang (2007): *The Effect of Restarts on the Efficiency of Clause Learning*. In: *proceedings of IJCAI*. pp. 2318–2323.

[21] F. Hutter, D. Tompkins & H. Hoos (2002): *Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT*. In: *proceedings of CP*. pp. 233–248.

[22] N. Jussien & O. Lhomme (2000): *Local Search with Constraint Propagation and Conflict-Based Heuristics*. In: *AAAI/IAAI*. pp. 169–174.

[23] F. Letombe & J. Marques-Silva (2008): *Improvements to Hybrid Incremental SAT Algorithms*. In: *proceedings of SAT*. pp. 168–181.

[24] C.M. Li, W. Wei & H. Zhang (2007): *Combining Adaptive Noise and Look-Ahead in Local Search for SAT*. In: *proceedings of SAT*. pp. 121–133.

[25] B. Mazure, L. Saïs & E. Grégoire (1995): *TWSAT : a new local search algorithm for SAT : performance and analysis*. In: *Proceedngs of the Workshop CP95 on Solving Really Hard Problems*. pp. 127–130.

[26] B. Mazure, L. Saïs & E. Grégoire (1998): *Boosting Complete Techniques Thanks to Local Search Methods*. *Ann. Math. Artif. Intell.* 22(3-4), pp. 319–331.

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang & S. Malik (2001): *Chaff: Engineering an Efficient SAT Solver*. In: *proceedings of DAC*. pp. 530–535.

[28] B. Selman & H. Kautz (1993): *An Empirical Study of Greedy Local Search for Satisfiability Testing*. In: *proceedings of AAAI*. pp. 46–51.

[29] B. Selman, H. Kautz & B. Cohen (1994): *Noise Strategies for Improving Local Search*. In: *proceedings of AAAI*. pp. 337–343.

[30] B. Selman, H. Kautz & D. McAllester (1997): *Ten Challenges in Propositional Reasoning and Search*. In: *proceedings of IJCAI*. pp. 50–54.

[31] L. Zhang, C. Madigan, M. Moskewicz & S. Malik (2001): *Efficient Conflict Driven Learning in Boolean Satisfiability Solver*. In: *proceedings of ICCAD*. pp. 279–285.